

CFD WITH OPENSOURCE SOFTWARE

A COURSE AT CHALMERS UNIVERSITY OF TECHNOLOGY
TAUGHT BY HÅKAN NILSSON

Project work:

A FSI tutorial on the axialTurbine tutorial case

Developed for FOAM-3.1-ext
Case files: axialTurbineFsiTut.tar.gz

Author:
Erik KARLSSON

Peer reviewed by:
NASER HAMEDI
HÅKAN NILSSON

Disclaimer: This is a student project work, done as part of a course where OpenFOAM and some other OpenSource software are introduced to the students. Any reader should be aware that it might not be free of errors. Still, it might be useful for someone who would like learn some details similar to the ones presented in the report and in the accompanying files. The material has gone through a review process. The role of the reviewer is to go through the tutorial and make sure that it works, that it is possible to follow, and to some extent correct the writing. The reviewer has no responsibility for the contents.

January 18, 2015

Contents

1	Introduction	2
2	Pre-processing	3
2.1	Getting started	3
2.2	fsiFoam	3
2.3	Necessary geometry modifications	4
2.4	The blockMeshDict	5
2.4.1	Fluid	5
2.4.2	Solid	11
2.5	Important files	12
2.5.1	Fluid	13
2.5.2	Solid	14
2.6	Boundary conditions	15
3	Running the case	16
4	Post-processing in ParaView	16

1 Introduction

This tutorial describes how to pre-process, run and post-process a case involving a fluid structure interaction (FSI). It also contains a description of necessary modifications to the geometry, figure (1) and (2), and a short description of special boundary conditions. This tutorial has no focus on the physics of the case.

The geometry of the axialTurbine case in the simpleSRFFoam solver consists of a solid region and a fluid region. The solid region consists of a turbine blade with axial length, $l_{ax} = 0.1m$, radial length, $l_r = 0.05m$, and thickness at the hub, $h_{hub} = 0.005m$ (figure 2). The fluid region is defined as a fifth of the total domain, the tangential shift between the cyclicGgi-patches are hence 72 degrees (2). There are inlet and outlet regions upstream and downstream of the blade which are both $0.02m$.

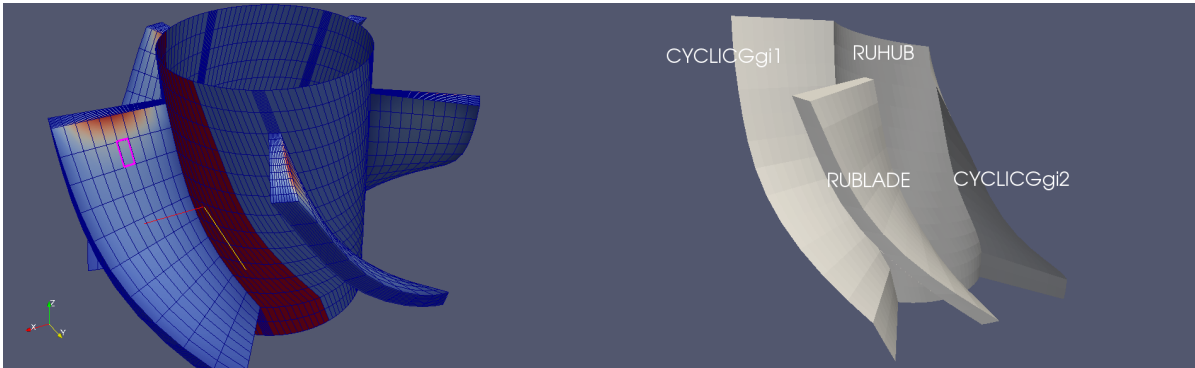


Figure 1: Geometry of the entire axialTurbineFsiTut tutorial case domain.

Figure 2: Geometry of the meshed domain in the axialTurbineFsiTut tutorial case.

2 Pre-processing

This section covers the necessary setup needed to get the axialTurbine case running with the solver fsiFoam.

2.1 Getting started

The FSI package is not included in OpenFoam 3.1-extend, it has to be downloaded:

http://openfoamwiki.net/index.php/Extend-bazaar/Toolkits/Fluid-structure_interaction

Then click the link `File:Fsi31.tar.gz`, where the current version can be downloaded.

Extract and install by:

```
run
mv *downloadDir*/Fsi_31.tar.gz .
tar -xzf Fsi_31.tar.gz
rm Fsi_31.tar.gz
cd FluidStructureInteraction/
cd src/
./Allwmake
```

The application of our interest is `fsiFoam`.

2.2 fsiFoam

The solver `fsiFoam` is a partitioned, strongly coupled FSI solver. The partitioned method is to separately solve the governing equations of the flow and the structure, with two independent solvers. However the whole FSI simulation takes the settings of the IO and time control that are specified in the fluid solver `system/controlDict`. The exchanged variables from the solid solver to the fluid solver are displacement increment and velocity.

The general `fsiFoam` algorithm is to:

1. Move fluid and solid mesh
2. Solve fluid equations
3. Solve solid equations
4. Compute residual

Iterate 1-4 until residual is smaller than the specified tolerance. The exchanged variables are pressure, p_Γ , and viscous force, \vec{t}_Γ in the fluid. In the structure, displacement increment, \vec{u}_Γ , and velocity, \vec{v}_Γ , are exchanged.

5. Increment time

The file structure of a `fsiFoam` case can be found in `FluidStructureInteraction` tutorials.

```
cd $FOAM_RUN/FluidStructureInteraction/run/fsiFoam/
```

The case directory has a `fluid` and `solid` folder, as well as the files `makeLinks`, `makeSerialLinks` and `removeLinks`. The folder, `fluid`, contains the files required for the fluid computation. Similarly does the folder, `solid`, contain the files for the solid computation. Since the case is run from the fluid directory the solid directory has to be linked to fluid directory. This is done by running the

files `makeLinks` or `makeSerialLinks`, for parallel or serial computation respectively. Running the file `removeLinks` will remove the links between the directories.

To get started with the setup, an existing tutorial can be copied:

```
run
cp -rp FluidStructureInteraction/run/fsiFoam/3dTube/ .
mv 3dTube axialTurbineFsiTut/
cd axialTurbineFsiTut/fluid/
```

Remove some of the unnecessary files and copy the axialTurbine geometry to `constant/polyMesh`:

```
rm createZones
rm gpDeflection
rm time-series
cd constant/polyMesh
rm blockMeshDict
cp $FOAM_TUTORIALS/incompressible/simpleSRFFoam/axialTurbine/constant/polyMesh/blockMeshDict.m4 .
```

If wanted all files can be kept except `blockMeshDict`, which has to be removed. The `createZones` commands will be included in the `Allrun` script, the `gpDeflection` files is used for plotting which will be done in `paraView` instead, `time-series` is used to vary the initial conditions which wont be done in this tutorial.

2.3 Necessary geometry modifications

The geometry of this tutorial is based on is the `axialTurbine` (figure 3) tutorial in:

```
$FOAM_TUTORIALS/incompressible/simpleSRFFoam/axialTurbine/constant/polyMesh
```

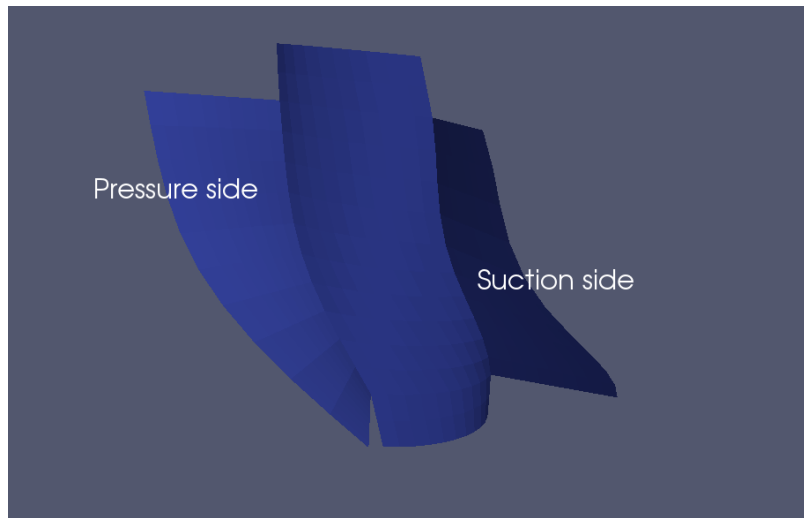


Figure 3: Original geometry

This geometry is however not suitable for FSI simulations. In the original geometry (figure 3) the blade suction and pressure side is modeled at the opposite sides of the domain. For FSI simulations the blade needs fluid on all interaction interfaces, it also needs a thickness to have any stiffness and to be able to mesh the solid region. To achieve this, the blade has to be moved to the middle of the domain and a thickness has to be added.

The already modified files and the final `blockMeshDict` can be found in the accompanying files.

Here follows a step-by-step tutorial in how to modify the existing files.
 To get started with the setup an existing tutorial can be copied:

```
run
cp -rp FluidStructureInteraction/run/fsiFoam/3dTube/ .
mv 3dTube axialTurbineFsiTut/
cd axialTurbineFsiTut/fluid/
```

Remove some of the unnecessary files and copy the axialTurbine geometry to `constant/polyMesh`:

```
rm createZones
rm gpDeflection
rm time-series
cd constant/polyMesh
rm blockMeshDict
cp $FOAM_TUTORIALS/incompressible/simpleSRFFoam/axialTurbine/constant/polyMesh/blockMeshDict.m4 .
```

2.4 The blockMeshDict

The mesh is created with `m4` and `blockMesh`. Cylindrical coordinates, with modified angle: $1/20$, are utilized. A modified angle is used as it results in a better mesh quality when the coordinates are transformed back to cartesian. In this section the necessary changes in the fluid side `blockMeshDict.m4` are described. Care has to be taken in order to not change the tangential length of the domain.

2.4.1 Fluid

Firstly a blade thickness of 0.005m is defined by adding `define(h, 0.005/hr/20)` after the hub- and shroud radial definitions. Now the vertices all along the pressure side boundary (see figure 3) are moved towards the suction side with half the tangential domain length. This is done in order to later be able add fluid to the right of the suction side and still keep the total tangential domain length at 72 degrees. The vertices defining the blade suction side are moved towards the pressure side to make room for a blade thickness.

```
vertices //(radial [m], tangential [radians], axial [m])
(
//Runner hub:
    (hr calc(RUt0+0.5*RUp) RUa0) vlabel(RU01b)
    (hr calc(RUt0+RUp-h) RUa0) vlabel(RU0rb)
    (hr calc(RUt1+0.5*RUp) calc(RUa0+RUoa1)) vlabel(RU11b)
    (hr calc(RUt1+RUp-h) calc(RUa0+RUoa1)) vlabel(RU1rb)
    (hr calc(RUt2+0.5*RUp) calc(RUa0+RUoa1+RUa1)) vlabel(RU21b)
    (hr calc(RUt2+RUp-h) calc(RUa0+RUoa1+RUa1)) vlabel(RU2rb)
    (hr calc(RUt3+0.5*RUp) calc(RUa0+RUoa1+RUa1+RUia1)) vlabel(RU31b)
    (hr calc(RUt3+RUp-h) calc(RUa0+RUoa1+RUa1+RUia1)) vlabel(RU3rb)

//Runner shroud:
    (sr calc(RUt0+0.5*RUp) RUa0) vlabel(RU01t)
    (sr calc(RUt0+RUp-h) RUa0) vlabel(RU0rt)
    (sr calc(RUt1+0.5*RUp) calc(RUa0+RUoa1)) vlabel(RU11t)
    (sr calc(RUt1+RUp-h) calc(RUa0+RUoa1)) vlabel(RU1rt)
    (sr calc(RUt2+0.5*RUp) calc(RUa0+RUoa1+RUa1)) vlabel(RU21t)
    (sr calc(RUt2+RUp-h) calc(RUa0+RUoa1+RUa1)) vlabel(RU2rt)
    (sr calc(RUt3+0.5*RUp) calc(RUa0+RUoa1+RUa1+RUia1)) vlabel(RU31t)
    (sr calc(RUt3+RUp-h) calc(RUa0+RUoa1+RUa1+RUia1)) vlabel(RU3rt)
```

Additional vertices have to be added in order to create blocks upstream and downstream the blade now that the blade has a thickness. This is done by adding

```
(hr calc(RUt0+RUp) RUa0) vlabel(extraRU0rb)
(hr calc(RUt1+RUp) calc(RUa0+RUoal)) vlabel(extraRU1rb)
(sr calc(RUt0+RUp) RUa0) vlabel(extraRU0rt)
(sr calc(RUt1+RUp) calc(RUa0+RUoal)) vlabel(extraRU1rt)
(hr calc(RUt2+RUp) calc(RUa0+RUoal+RUal)) vlabel(extraRU2rb)
(hr calc(RUt3+RUp) calc(RUa0+RUoal+RUal+RUial)) vlabel(extraRU3rb)
(sr calc(RUt2+RUp) calc(RUa0+RUoal+RUal)) vlabel(extraRU2rt)
(sr calc(RUt3+RUp) calc(RUa0+RUoal+RUal+RUial)) vlabel(extraRU3rt)
```

Vertices for the new fluid domain to the right of the blade is added by:

```
(hr calc(RUt0+1.5*RUp) RUa0) vlabel(rExtraRU0rb)
(hr calc(RUt1+1.5*RUp) calc(RUa0+RUoal)) vlabel(rExtraRU1rb)
(sr calc(RUt0+1.5*RUp) RUa0) vlabel(rExtraRU0rt)
(sr calc(RUt1+1.5*RUp) calc(RUa0+RUoal)) vlabel(rExtraRU1rt)
(hr calc(RUt2+1.5*RUp) calc(RUa0+RUoal+RUal)) vlabel(rExtraRU2rb)
(hr calc(RUt3+1.5*RUp) calc(RUa0+RUoal+RUal+RUial)) vlabel(rExtraRU3rb)
(sr calc(RUt2+1.5*RUp) calc(RUa0+RUoal+RUal)) vlabel(rExtraRU2rt)
(sr calc(RUt3+1.5*RUp) calc(RUa0+RUoal+RUal+RUial)) vlabel(rExtraRU3rt)
```

);

After defining the new vertices five blocks should be added; upstream and downstream of the blade and three blocks on the new pressure side. Under blocks add:

```
hex2D(RUOr, extraRU0r, extraRU1r, RU1r) //Block downstream blade
rotor
(RUtc RUoac RUrc)
simpleGrading (1 1 1)

hex2D(RU2r, extraRU2r,extraRU3r,RU3r) //Block upstream blade
rotor
(RUtc RUoac RUrc)
simpleGrading (1 1 1)

hex2D(extraRU0r, rExtraRU0r, rExtraRU1r, extraRU1r) // Downstream block on pressure side
rotor
(RUtc RUoac RUrc)
simpleGrading (1 1 1)

hex2D(extraRU1r, rExtraRU1r, rExtraRU2r, extraRU2r) //Block intersecting with blade
rotor
(RUtc RUBac RUrc)
simpleGrading (1 1 1)

hex2D(extraRU2r, rExtraRU2r, rExtraRU3r, extraRU3r) // Upstream block on pressure side
rotor
(RUtc RUiac RUrc)
simpleGrading (1 1 1)
```

To create a curved blade splines are used, these have to be added due to the relocation of the blade. The syntax for spline is:

```
spline startVertex endVertex (
(list of interpolation points)
)
```

The same set of splines, as in the original geometry, are used. They have to be slightly modified since the position of the outer boundary has moved in the tangential direction. This is done by modifying the θ -coordinate (the RUp entry):

```
edges
(
//Runner
  spline RU1lt RU2lt // Modified due to altering outer boundary
  (
    (sr calc(RUt1+0.5*RUp+0.65*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal))
  )
  spline RU1lb RU2lb
  (
    (hr calc(RUt1+0.5*RUp+0.65*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal))
  )
  spline RU1rt RU2rt // Due to blade in domain center
  (
    (sr calc(RUt1+RUp+0.75*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal))
  )
  spline RU1rb RU2rb
  (
    (hr calc(RUt1+RUp+0.75*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal))
  )

  spline extraRU1rt extraRU2rt
  (
    (sr calc(RUt1+RUp+0.65*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal))
  )
  spline extraRU1rb extraRU2rb
  (
    (hr calc(RUt1+RUp+0.65*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal))
  )
  spline rExtraRU1rt rExtraRU2rt // Splines on outer boundary, pressure side
  (
    (sr calc(RUt1+1.5*RUp+0.65*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal))
  )
  spline rExtraRU1rb rExtraRU2rb
  (
    (hr calc(RUt1+1.5*RUp+0.65*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal))
  )
);
```


When creating patches under **boundary**, all faces on the interaction interface have to be in the same patch. Inlet and outlet are of type patch, the interaction interface type is wall. On the outer boundaries in the tangential direction, the boundary condition cyclicGgi is set.

```
boundary
(
  RUINLET
  {
    type          patch;
    faces
    (
      quad2D(RU3r, RU3l)
      quad2D(extraRU3r,RU3r)
      quad2D(rExtraRU3r, extraRU3r)
    );
  }

  RUOUTLET
  {
    type          patch;
    faces
    (
      quad2D(RU0l, RU0r)
      quad2D(RU0r, extraRU0r)
      quad2D(extraRU0r, rExtraRU0r)
    );
  }

  RUBLADE
  {
    type wall;
    faces
    (
      quad2D(RU1r, RU2r) //Blade left
      quad2D(extraRU1r, RU1r) //wall bottom
      quad2D(RU2r, extraRU2r) //Wall top
      quad2D(extraRU2r, extraRU1r) //Blade right
    );
  }
}
```

```

RUCYCLIC1
{
    type            cyclicGgi;
    shadowPatch     RUCYCLIC2;
    zone            RUCYCLIC1Zone;
    bridgeOverlap   false;
    rotationAxis    (0 0 1);
    rotationAngle   72;
    separationOffset (0 0 0);
    faces
    (
        quad2D(RU11, RU01)
        quad2D(RU21, RU11)
        quad2D(RU31, RU21)
    );
}

RUCYCLIC2
{
    type            cyclicGgi;
    shadowPatch     RUCYCLIC1;
    zone            RUCYCLIC2Zone;
    bridgeOverlap   false;
    rotationAxis    (0 0 1);
    rotationAngle   -72;
    separationOffset (0 0 0);
    faces
    (
        quad2D(rExtraRU0r, rExtraRU1r)
        quad2D(rExtraRU1r, rExtraRU2r)
        quad2D(rExtraRU2r, rExtraRU3r)
    );
}

```

While using the cyclicGgi BC the shadowPatch is specified as the opposite outer boundary (see figure (2)). A zone has to be specified, which will be discussed later. The `bridgeOverlap` option can, in RUCYCLIC1 and RUCYCLIC2, be set to either `True` or `false`. Use `True`, with caution, if there are any uncovered faces. Due to discretization, it's not always possible to avoid uncovered faces. In our case all faces are covered at a 72 degree rotation angle, the option is set to `false`. `rotationAxis` simply specifies around which axis to rotate. `rotationAngle` specifies where the shadowPatch is located in relation to the masterPatch. For the boundary CYCLICGgi2 (figure (2)) rotation angle is negative.

GGI is used to couple interfaces in the mesh where the nodes on each side of the interface do not match. Weight factors are used to decide how much information should be transferred from one side of the ggi to its neighbouring cells on the other side. GGI is, in our case, probably unnecessary but is used since it used in the original geometry.

The turbine hub and shroud are type `wall`.

```
RUHUB
{
    type wall;
    faces
    (
        backQuad(RU01, RU0r, RU1r, RU11)
        backQuad(RU0r, extraRU0r, extraRU1r, RU1r)
        backQuad(RU11, RU1r, RU2r, RU21)
        backQuad(RU21, RU2r, RU3r, RU31)
        backQuad(RU2r, extraRU2r, extraRU3r, RU3r)
        backQuad(extraRU0r, rExtraRU0r, rExtraRU1r, extraRU1r)
        backQuad(extraRU1r, rExtraRU1r, rExtraRU2r, extraRU2r)
        backQuad(extraRU2r, rExtraRU2r, rExtraRU3r, extraRU3r)
    );
}

RUSHROUD
{
    type wall;
    faces
    (
        frontQuad(RU01, RU0r, RU1r, RU11)
        frontQuad(RU0r, extraRU0r, extraRU1r, RU1r)
        frontQuad(RU11, RU1r, RU2r, RU21)
        frontQuad(RU21, RU2r, RU3r, RU31)
        frontQuad(RU2r, extraRU2r, extraRU3r, RU3r)
        frontQuad(extraRU0r, rExtraRU0r, rExtraRU1r, extraRU1r)
        frontQuad(extraRU1r, rExtraRU1r, rExtraRU2r, extraRU2r)
        frontQuad(extraRU2r, rExtraRU2r, rExtraRU3r, extraRU3r)
    );
}
);
```

This concludes the necessary modifications to the fluid side `blockMeshDict`, however the solid has to be meshed as well.

2.4.2 Solid

The same original geometry as for the fluid was used when creating the solid.

`$FOAM_TUTORIALS/incopressible/simpleSRFFoam/axialTurbine/constant/polyMesh`

The solid region is a less complex geometry, consisting of only one block. The initial definitions in the solid side `blockMeshDict` are the same as for the fluid side. The changes to `blockMeshDict.m4` starts with the vertices.

```
vertices //(radial [m], tangential [radians], axial [m])
(
//Runner hub:
    (hr calc(RUt1+RUp-h) calc(RUa0+RUoal)) vlabel(RU1lb)
    (hr calc(RUt1+RUp) calc(RUa0+RUoal)) vlabel(RU1rb)
    (hr calc(RUt2+RUp-h) calc(RUa0+RUoal+RUal)) vlabel(RU2lb)
    (hr calc(RUt2+RUp) calc(RUa0+RUoal+RUal)) vlabel(RU2rb)

//Runner shroud:
    (sr calc(RUt1+RUp-h) calc(RUa0+RUoal)) vlabel(RU1lt)
    (sr calc(RUt1+RUp) calc(RUa0+RUoal)) vlabel(RU1rt)
    (sr calc(RUt2+RUp-h) calc(RUa0+RUoal+RUal)) vlabel(RU2lt)
    (sr calc(RUt2+RUp) calc(RUa0+RUoal+RUal)) vlabel(RU2rt)
);
```

The vertices are placed to match the gap in the fluid mesh. Only one block is needed.

```
blocks
(
    hex2D(RU1l, RU1r, RU2r, RU2l)
    rotor
    (RUtc RUBac URrc)
    simpleGrading (1 1 1)
);
```

Splines are used to match the curvature of the gap in the fluid mesh. The original splines has to be modified, as for the fluid region, due to new tangential location of the boundary.

```
edges // Inappropriate with arc due to coordinate conversion
(
//Runner
    spline RU1lt RU2lt
    (
        (sr calc(RUt1+RUp-h+0.65*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal))
    )
    spline RU1lb RU2lb
    (
        (hr calc(RUt1+RUp-h+0.65*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal))
    )
    spline RU1rt RU2rt
    (
        (sr calc(RUt1+RUp+0.65*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal))
    )
    spline RU1rb RU2rb
    (
        (hr calc(RUt1+RUp+0.65*(RUt2-(RUt1))) calc(RUa0+RUoal+0.5*RUal))
    )
);
```

When defining the patches, the entire solid FSI interface has to be in the same patch. The blade is fixed at the inner and outer radius. Type `wall` is used for all solid boundaries.

```
boundary
(
  RUBLADEsolid
  {
    type          wall;
    faces
    (
      quad2D(RU1r, RU2r) //Blade_Right
      quad2D(RU2l, RU1l) //Blade_Left
      quad2D(RU1l, RU1r) //Blade_bottom
      quad2D(RU2r, RU2l) //Blade_top
    );
  }

  RUHUB
  {
    type          wall;
    faces
    (
      backQuad(RU1l, RU1r, RU2r, RU2l)
    );
  }

  RUSHROUD
  {
    type          wall;
    faces
    (
      frontQuad(RU1l, RU1r, RU2r, RU2l)
    );
  }
);
```

2.5 Important files

The file `makeSerialLinks` creates links between the fluid- and solid directories and copies the `0_orig` to a `0` directory. When running the script the positional parameter `$1` contains the first argument and `$2` the second argument.

```
#!/bin/tcsh

cd $1
cp -r 0_orig 0
cd constant
ln -s ../../$2/constant solid
cd ../system
ln -s ../../$2/system solid
cd ../0
ln -s ../../$2/0 solid
cd ../../
```

The file `removeLinks` is used to remove the links.

2.5.1 Fluid

A dynamic mesh is used to move the FSI interface. The internal grid points in the fluid mesh adjust their positions when the FSI interface moves. Settings for the dynamic mesh is found in `fluid/constant/dynamicMeshDict`:

```
dynamicFvMesh    dynamicMotionSolverFvMesh;

solver velocityLaplacian; //Solve the laplacian equation
diffusivity quadratic inverseDistance 1(RUBLADE);
```

The one (1) specifies the number of patches, in our case just one, then the patch name of the fluid side FSI interface is specified.

In the file `fluid/constant/flowProperties` the choice of flow model is specified. The model used is `consistentIcoFlow` which is equivalent to `icoDyMFoam` solver with consistent `ddtPhiCorr`. In `fluid/constant` the file `fsiProperties` contains settings for coupling the solvers.

```
solidPatch RUBLADEsolid;
solidZone RUBLADEsolidZone;

fluidPatch RUBLADE;
fluidZone RUBLADEZone;

...

//couplingScheme FixedRelaxation;
couplingScheme Aitken;

//couplingScheme IQN-ILS;
couplingReuse 0;

coupled yes;
```

Here are the FSI interaction interfaces specified as well as coupling scheme. The coupling process requires information of FSI zones. These zones are created the same way by both fluid and solid solver. Take the fluid solver for example. In the `fluid/` folder there is a file, `setBatch` which should save the following content:

```
faceSet RUCYCLIC1Zone new patchToFace RUCYCLIC1
faceSet RUCYCLIC2Zone new patchToFace RUCYCLIC2
faceSet RUBLADEZone new patchToFace RUBLADE
quit
```

Where the first two lines specifies zones for the GGI interpolation. The third line specifies the zone needed for the solver coupling. Running the `Allrun` script then creates the zones. The `Allrun` script also creates the `blockMeshDict` from the `.m4` file, runs `blockMesh`, scales the tangential coordinate and transforms the mesh to Cartesian coordinates using `transformPoints`.

```
#!/bin/bash
# Source tutorial run functions
. $WM_PROJECT_DIR/bin/tools/RunFunctions

# Get application from system/controlDict
application=`getApplication`

# Create the blockMeshDict-file
```

```

m4 < constant/polyMesh/blockMeshDict.m4 > constant/polyMesh/blockMeshDict

# Runns blockMesh
runApplication blockMesh

# Transform to cartesian coordinates
transformPoints -scale "(1 20 1)"
transformPoints -cylToCart "((0 0 0) (0 0 1) (1 0 0))"

# Set 0-directory and create GGI and coupling set:
runApplication setSet -batch setBatch
runApplication setsToZones -noFlipMap

```

2.5.2 Solid

Now lets have a look at the important files for the solid computation.

```
cd $FOAM_RUN/axialTurbineFsiTut/solid/constant/
```

In the file `/solid/constant/stressProperties` the stress model is specified. In this case `unsTotalLagrangianStress` is used. It is a large strain, elastic stress, analysis solver based on total Lagrangian displacement formulation.¹ The file `/solid/constant/rheologyProperties` specifies the density, Young's modulus and Poisson's ratio.

```

planeStress no; //Yes for 2D, no for 3D
rheology
{
    type linearElastic;
    rho rho [1 -3 0 0 0 0 0] 1000;
    E E [1 -1 -2 0 0 0 0] 5.0e6;
    nu nu [0 0 0 0 0 0 0] 0.4;
}

```

As for the fluid, the `axialTurbineFsiTut/solid` should contain a file, `setBatch`, which contains information about the FSI interaction zone.

```

faceSet RUBLADEsolidZone new patchToFace RUBLADEsolid
quit

```

¹OpenFOAM Library for Fluid Structure Interaction, 9th OpenFOAM Workshop - Zagreb, Croatia

An Allrun script, where the mesh is created, transformed and the zone is created, is needed. It has the same functionalities as the fluid/Allrun script and should contain the following code:

```
#!/bin/bash
# Source tutorial run functions
. $WM_PROJECT_DIR/bin/tools/RunFunctions

# Create the blockMeshDict file
m4 < constant/polyMesh/blockMeshDict.m4 > constant/polyMesh/blockMeshDict

# Run blockMesh
runApplication blockMesh

# Transform the mesh to cartesian coordinates and rescale the angle
transformPoints -scale "(1 20 1)"
transformPoints -cylToCart "((0 0 0) (0 0 1) (1 0 0))"

# Set 0-directory and create GGI set:
runApplication setSet -batch setBatch
runApplication setsToZones -noFlipMap
```

2.6 Boundary conditions

For the fluid computation on the FSI interface, the boundary condition `movingWallVelocity` is set. This BC corrects the flux due to mesh motion in such a way that the total flux through the moving wall is zero.

```
RUBLADE
{
    type            movingWallVelocity;
    value           uniform (0 0 0);
}
```

For the solid FSI interface in `solid/0/D` the boundary condition `tractionDisplacement` is used. Here externally imposed pressure, traction and displacement are specified.

```
RUBLADEsolid
{
    type            tractionDisplacement;
    traction        uniform (0 0 0);
    pressure         uniform 0;
    value           uniform (0 0 0);
}
```

We have the initial conditions according to table (1).

Table 1: Initial conditions for the axialTurbineFsiTut tutorial

Variable	Initial conditions
p	internalField uniform 0, walls zeroGradient, outlet uniform 0
$pointMotionU$	internalField uniform (0 0 0), all uniform (0 0 0)
U	internalField uniform (0 0 0), walls uniform (0 0 0), inlet uniform (0 0 -0.5), outlet zeroGradient
D	internalField uniform (0 0 0), walls uniform (0 0 0)
$pointD$	internalField uniform (0 0 0), walls uniform (0 0 0)

3 Running the case

First start with creating the mesh and required zones for the solid computation.

```
run
cd axialTurbinFsiTut/solid
./Allrun
```

Then links between the fluid- and solid directories have to be created.

```
cd ..
./makeSerialLinks fluid solid
```

When this is done, the mesh and zones for the fluid computation should be created.

```
cd fluid
./Allrun
```

Now start the computation with:

```
fsiFoam >&log.fsiFoam &
```

Make sure the simulation is running with:

```
tailf log.fsiFoam
```

Use `ctrl+c` to exit `tailf`

4 Post-processing in ParaView

Load both the fluid and solid domain into ParaView with the command `paraFoam` while in the `fluid` directory. Since only a fifth of the domain is modeled, a method for visualizing the entire domain is needed. For this the ParaView filter, `transform` is used. Use `transform` filter four times on the original object, and rotate around the z-axis with rotation angle 72, 144, -72, -144 degrees to show the entire domain (see figure 4). To clearly see the blades, untick the box `internalMesh` and tick the box `RUHUB` instead, figure (5). In order to not have to redo this procedure every time all blades are to be visualized, the geometry with all transformations can be saved as a state. This is done by `File` → `Save State`. Then use `$ paraview --state=*stateName*.pvsm` to load the state next time the case is to be visualized in paraview.

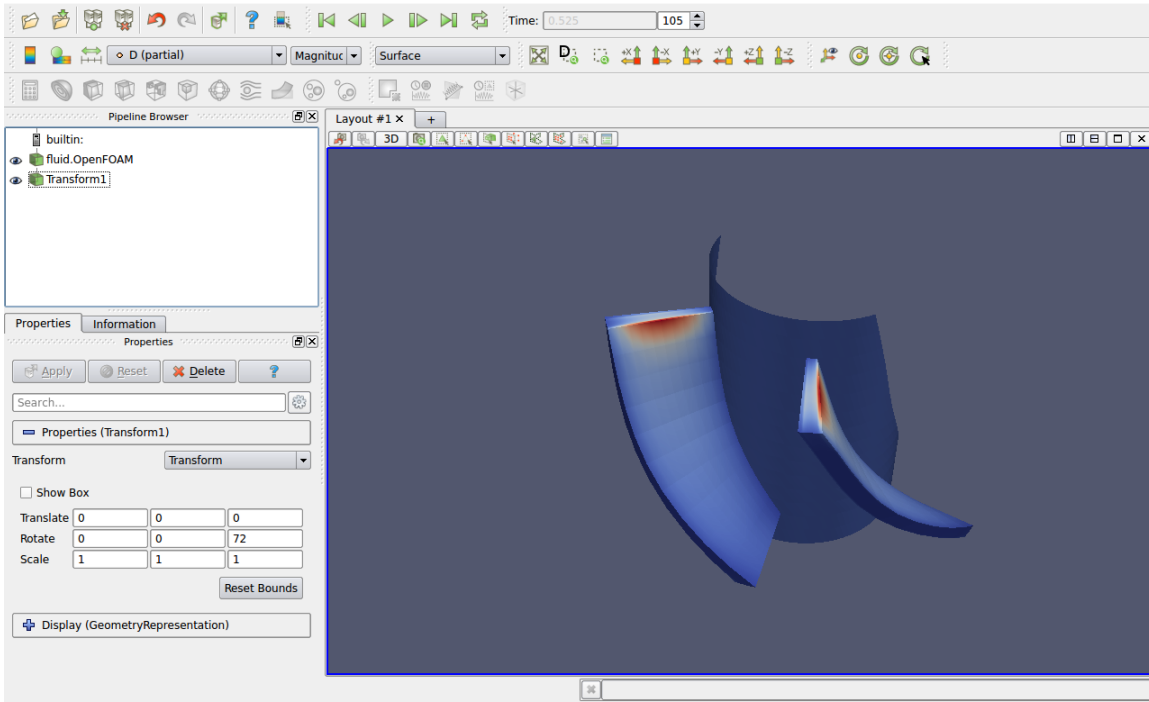


Figure 4: Initial geometry and transform-filter with 72 degree rotation

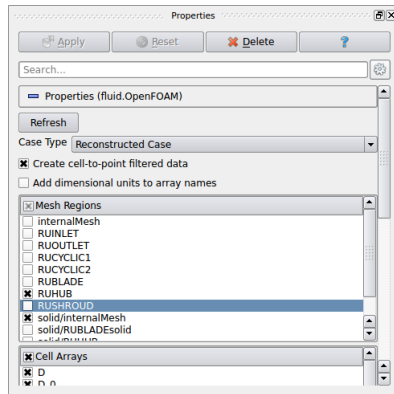


Figure 5: Untick internalMesh to see blade

Results from the simulation is shown in figures (6) and (7). A spike in the deflection value occur before the flow is fully developed.

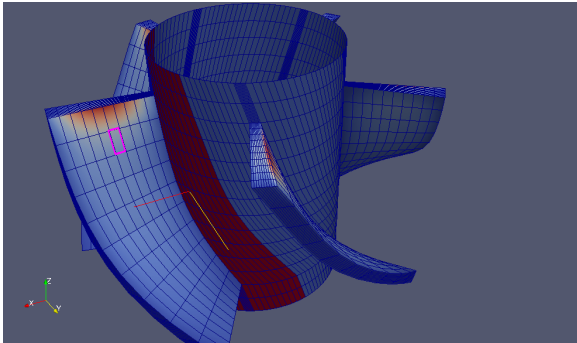


Figure 6: Colored by deflection

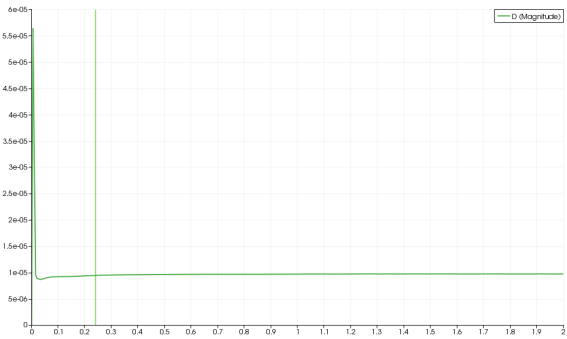


Figure 7: Deflection over time